CS 1358 Introduction to Programming in Python

Fall Semester 2023

Prof. Pai H. Chou

Self-Check 12

Answer the following questions to check your understanding of your material. Expect the same kind of questions to show up on your tests.

## 1. Definitions and Short Answers - functions

1. Given a for loop:  
    1 for i in L:  
    2 print(i)  
   Can L be the following? If so, what does the loop print? If not, why not?
   1. ['a', 'b', 'c']
   2. ('a', 'b', 'c')
   3. 'abc'
   4. {'a', 'b', 'c'}
   5. {'a': 100, 'b': 200, 'c': 300}
   6. 0xabcd
   7. range(3)
   8. 23+4j
2. Given an iterable data structure L,
   1. How do you obtain an **iterator** r of L?
   2. Once you have an iterator r, what can you do to get the next value?
   3. What happens when you call next(r) but your iterator r has finished iterating over all values of L?
   4. Is there a limit to the number of iterators that you can create on the same iterable?
3. Assume you have   
    1 D = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']  
    2 r = iter(D)  
    3 L = [next(r) for i in range(3)]  
    4 s = iter(D)  
    5 M = [next(s) for i in range(2)]  
   after executing these five lines
   1. What is the value of L?
   2. What is the value of M?
   3. What is the value of D?
4. Recall the Vector class from the previous lecture,  
    1 import operator as op   
    2 class Vector:   
    3 def \_\_init\_\_(self, \*v):   
    4 self.\_v = list(v) # covert tuple to list   
    5 def \_\_repr\_\_(self):   
    6 return \_\_class\_\_.\_\_name\_\_+repr(tuple(self.\_v))   
     
   Suppose a class defines an \_\_iter\_\_() special method, and v is an instance of Vector.
   1. How does Python intend that v's \_\_iter\_\_() special method be invoked by the programmer? Hint: not v.\_\_iter\_\_()
   2. What kind of object should the \_\_iter\_\_() method return?
   3. What is one simple way to implement Vector's \_\_iter\_\_() method, given that the iterator for Vector would essentially be the same as the iterator for the list self.\_v ?
5. An alternative to part 4.(c) is to define a class for VectorIterator, and Vector's \_\_iter\_\_() method would instantiate and return it. The code is as follows:  
    1 class Vector:  
    2 def \_\_iter\_\_(self):  
    3 return Vector\_Iterator(self):  
    4 ....  
    5 class VectorIterator:  
    6 def \_\_init\_\_(self, vec):  
    7 self.\_vec = vec  
    8 self.\_i = 0  
    9 def \_\_next\_\_(self):  
   10 if self.\_i >= len(self.\_vec):  
   11 raise StopIteration  
   12 val = self.\_vec[self.\_i]  
   13 self.\_i += 1  
   14 return val  
   1. In VectorIterator's constructor, what is the purpose of initializing \_i = 0?
   2. Why does VectorIterator's constructor need to set its \_vec attribute to the iterable? Why isn't it enough to just keep track of its position \_i?
   3. How does Python intend that the \_\_next\_\_() method of a VectorIterator instance vi be invoked? Hint: not vi.\_\_next\_\_()
   4. How does \_\_next\_\_() special method indicate that it has finished iterating all elements?
6. Assume Vector is iterable, rewrite the following for-loop using a while loop and explicit iter() instantiation, next(), and catching StopIteration exception:  
    1 v = Vector(7, 1, 4, 3, 9, 6, 5)  
    2 for i in v:  
    3 print(i, end='')
7. Can any iterable object v be passed as arguments to
   1. list(v)
   2. max(v)
8. For the Blackjack game example, Card is declared as a class:   
    1 class Card:  
    2 ACE, JACK, QUEEN, KING = 'A', 'J', 'Q', 'K'   
    3 FACES = (ACE,2,3,4,5,6,7,8,9,10, JACK, QUEEN, KING)  
    4 SUITS = tuple(map(chr, (9824, 9827, 9829, 9830)))   
    5 SPADE, CLUB, HEART, DIAMOND = SUITS # ♠ ♣ ♥ ♦  
    6 def \_\_init\_\_(self, suit, face):    
    7 self.\_suit = suit   
    8 self.\_face = face   
    9 def \_\_int\_\_(self):   
   10 if self.\_face in {Card.JACK,Card.QUEEN,Card.KING}:   
   11 return 10   
   12 return 1 if self.\_face == Card.ACE else self.\_face   
   13 def \_\_str\_\_(self):   
   14 return self.\_suit + str(self.\_face)   
   15 def \_\_repr\_\_(self):   
   16 return \_\_class\_\_.\_\_name\_\_ + \  
   17 repr((self.\_suit, self.\_face))
   1. Why is it a good practice to declare class attributes such as SPADE, CLUB, HEART, and DIAMOND even though Python3 handles unicode character literals such as '♠' '♣' '♥' '♦'.
   2. What is the purpose of special method \_\_int\_\_()?
   3. Why declare a \_\_str\_\_() special method even though \_\_repr\_\_() also exists and can make a string that represents the card?
   4. Is Card class **iterable**? Should it be iterable?
   5. Is Card class for instantiating **iterators**?
9. Continuing with the BlackJack example, a separate class named Deck is also declared.  
    1 class Deck:  
    2 def \_\_init\_\_(self):  
    3 self.\_deck = [Card(suit, face) \  
    4 for suit in Card.SUITS for face in Card.FACES]  
    5 def shuffle(self):  
    6 import random  
    7 random.shuffle(self.\_deck)  
    8 def \_\_iter\_\_(self):  
    9 return iter(self.\_deck)  
     
   1. Is Deck an iterable? If so, is it required to implement the \_\_getitem\_\_() special method?
   2. Explain how the Deck class is able to create iterators by simply returning iter(self.\_deck) from its \_\_iter\_\_() special method. Explain why this works.
10. In Single-player BlackJack,   
     1 def BlackJack():  
     2 D = Deck()  
     3 D.shuffle()  
     4 total = 0  
     5 it = iter(D)  
     6 while True:  
     7 c = next(it)  
     8 total += int(c)  
     9 print(f'your card: {c}, total = {total}.', end='')  
    10 if total > 21:  
    11 print(f'you lose! total = {total}')  
    12 break  
    13 if total == 21:  
    14 print(f'you win! total = 21')  
    15 break  
    16 ans = input('More cards? [y/n] ')  
    17 if ans not in 'Yy':  
    18 c = next(it) # draw one more to test  
    19 print(f'next card {c}. You ' +\  
    20 ('win' if total + c > 21 else 'lose'))  
    21 break  
    1. What kind of object is it as created on line 6?
    2. What kind of object is returned by a call to next(it) on line 7 or 18?
    3. Why doesn't this program have to handle the case where the iterator raises StopIteration exception when the deck is empty?
11. Is the following a **function** or a **generator**?
    1. def X(z):  
        for i in range(20):  
        yield i
    2. def Y(z):  
        for i in range(20):  
        return i
    3. def K(z):  
        for i in range(20):  
        yield i  
        return -1
12. if fib() is a generator for Fibonacci numbers, what is the syntax for
    1. instantiating a generator,
    2. generate the initial number,
    3. generate 10 more numbers after?  
       Fill in the blanks below.

g = \_\_\_\_ # instantiate generator  
init\_num = \_\_\_\_\_\_\_\_  
print('initial number = ', init\_num)  
for i in range(10):  
 num = \_\_\_\_\_\_  
 print(num)

1. Assume

g = fib() is a generator for Fibonacci numbers, and  
r = iter(deck) is an iterator where deck is an instance of iterable class Deck  
Which of the following are allowed?

* 1. list(r)
  2. list(g)
  3. list(fib())
  4. list(deck)
  5. [i for i in r]
  6. [i for i in g]
  7. [i for i in fib()]
  8. [i for i in iter(deck)]
  9. next(r)
  10. next(g)
  11. x, y, z = deck
  12. x, y, z = fib()
  13. x, y, z = g
  14. x, y, z = r

## 2. Programming

1. (Difficulty: ★★☆☆☆) Define a generator function CharRange, which generates a range of characters with inclusive bounds. It takes two parameters for the starting and ending characters. It yields one character at a time whose unicode number is one closer to the ending. For example,  
   $ python3 -i charrange.py  
   >>> cr = CharRange('A', 'E')  
   >>> list(cr)  
   ['A', 'B', 'C', 'D', 'E']  
   >>> dr = CharRange('E', 'A')  
   >>> list(dr)  
   ['E', 'D', 'C', 'B', 'A']  
   >>>   
     
   Hint: the generator looks like  
   def CharRange(start, end):  
    ...  
   It helps to convert between the character and the code using the ord() and chr() functions. To support stepping up or down, you need to check if the start is larger or smaller than the end. You may use range() to get one value at a time, but range() works for integers only; also, range's bound is exclusive, not inclusive, so you will need to make an adjustment for the bound's value. A generator uses yield instead of return to pass values back. After you finish yielding values, you don't have to do anything special, and your function will implicitly return None to mark the end of generation.
2. (Difficulty: ★★★☆☆) Define an iterable class named DaysInYear for iterating the days in a year. It takes the year as the argument to the constructor. Instead of implementing the \_\_iter\_\_ method to return the iterator object, it implements the \_\_getitem\_\_ method to return the ith value. The index to \_\_getitem\_\_ indicates the ith day of the year, where i = 0 means January 1, i = 1 means January 2, etc.  
     
   $ python3 -i daysinyear.py  
   >>> y = DaysInYear(2019)  
   >>> y[5]  
   '2019.01.06'  
   >>> y[364]  
   '2019.12.31'  
   >>> y[31]  
   '2019.02.01'  
   >>> y[365]  
   Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "daysinyear.py", line 15, in \_\_getitem\_\_

raise StopIteration

StopIteration

By defining the \_\_getitem\_\_ method, it makes the class iterable and you don't need to define the \_\_iter\_\_ method to return an iterator object -- the caller is responsible for tracking the iteration state. You do need to raise a StopIteration exception when the index is beyond the last day.   
This allows you to convert it to a list, use in a for loop, etc.

1. (Difficulty: ★★★★☆) Define a class named CountingList. It works like a list except it also keeps track of the number of times each element is accessed. It should also be iterable but its iterator outputs elements in decreasing order of access count. An access is defined by a call to either \_\_getitem\_\_(self, i) or \_\_setitem\_\_(self, i, val) where i may be an int or a slice.

>>> d = CountingList(('A', 'B', 'C', 'D', 'E'))  
>>> d[0], d[2], d[2], d[4] # these call \_\_getitem\_\_  
('A', 'C', 'C', 'E') # access counts = [1, 0, 2, 0, 1] >>> [d[-1], d[-2], d[4]]  
['E', 'D', 'E'] # access counts = [1, 0, 2, 1, 3]  
>>> for i in d:  
... print(i)  
...  
E  
C  
D  
A  
B

As you can probably figure out, you should define CountingList by subclassing from the built-in list class, like  
  
class CountingList(list):  
 def \_\_init\_\_(self, d = ()):  
 super().\_\_init\_\_(d)  
 # additional code here  
 def \_\_getitem\_\_(self, i):  
 # i is the index or slice.  
 # (1) use the same i to increment the access count,  
 # your code here...  
 # (2) return what the base class does, as below  
 return super().\_\_getitem\_\_(i)  
 def \_\_iter\_\_(self):  
 # This returns an iterator that outputs elements in   
 # order of decreasing access count.   
  
  
 need to implement the following methods:

* 1. The constructor:  
     It should first call the superclass's \_\_init\_\_ to initialize the list data structure, and then define additional data structures to keep an access count of the elements. A good one to use is another list structure, which can be indexed using the same index as that for accessing the CountingList's content. It contains the access count for the corresponding element in the CountingList and should be initialized to zero.
  2. The \_\_getitem\_\_(self, i) method:  
     It needs to intercept the accesses to each element by incrementing the corresponding count. Note that the type of i parameter can be either int or slice. In any case, this method needs to return the value, which can can be done by calling its base class's \_\_getitem\_\_ using the same i.
  3. Strictly speaking, you also need to intercept the \_\_setitem\_\_(self, i, val) method also. But you should make sure your \_\_getitem\_\_ works properly before you try the \_\_setitem\_\_. The idea is the same, since either one counts as one access.
  4. The \_\_iter\_\_(self) method:  
     It needs to return an iterator object but in order of decreasing access count. To do so, one way is to make a list whose elements are (access count, value) and sort in decreasing order, i.e., reverse=True. Then, you can return an iterator that iterates over the sorted value (but without the access count).